

# CS126: Introduction to Compilers

## Assignment 7: SEM2

**Out: 10/17/03**

**Due: 10/31/03**

### 1.0 Introduction

In the previous assignment you implemented the first two semantic passes. These found the types defined in the programs and then found and defined the methods and fields. They also implemented a number of semantic checks and cleaned up or normalized the abstract syntax trees. In this assignment you are going to complete the semantic analysis.

### 2.0 Requirements

The third and main semantic pass has seven primary goals:

1. Associating a definition with each Name node of the abstract syntax tree by looking up the name in the appropriate scope using appropriate parameters (e.g., ensuring that methods are looked up independently from other names). This involves determining the symbol referred to by the given Name node (taking into account any qualifiers, whether it is a call, argument types, etc.) and then calling `Name.setSymbol` with the appropriate symbol object.
2. Associating a type with each Expression node of the abstract syntax tree. This involves doing full name lookup and expression resolution and then determining the result type of each subexpression. The type is stored in the abstract syntax tree through a call to `Expression.setType` with the appropriate type object.
3. Editing the abstract syntax tree to insert operator nodes for any implicit casts that need to be performed. This involves noting where types need to be changed (say from char to int), and then calling  
`Expression.addCast(int, TargetOp op)`  
where the first argument indicates which child should be cast and the second is the cast operator. The latter should be one of `OP_CHAR_TO_INT` or `OP_INT_TO_CHAR`.
4. Editing the abstract syntax tree to normalize calls and field access. Static calls should end up with a Name node without a qualifier; non-static calls should have a Name node with a qualifier that will be the first argument to be passed in (nominally 'this'); fields will have a qualifier that is the object being referenced.
5. Determining the target operator for each source operator in an `OpExpression` node. This is done by calling

`OpExpression.setTargetOperator(TargetOp op)`  
with the appropriate target operator. The target operators are listed in `DecafConstants.java`.

6. Generating any errors that arise while attempting to look up names or resolve expressions using meaningful error messages.
7. Doing any remaining semantic checking required by the language definition.

To determine the appropriate semantic checks you should go through the DECAF semantics handout line by line and check that each of the constraints specified in that manual is implemented in either the parser or one of your semantic passes. Some of the things that you will have to check include:

- Formals and locals cannot have type `void`.
- Access restrictions.
- `Super` and `this` cannot be used in a static method.
- You can only assign to lvalue type expressions (note that we provide a `Expression.isLvalue()` method for this purpose).
- `New` with dimensions can only be applied to certain primitive types; `new` without dimensions can only be applied to objects.
- Non-static methods cannot be called from a static context.
- All classes used are defined.

### 3.0 Creating Your Class

You should create a visitor class to manage this third pass as you did with the first two passes.

You should test your pass as before, giving it both correct and incorrect programs and printing out the abstract syntax tree that results (as well as any error messages). You should hand-check the abstract syntax trees as much as possible to ensure that they are logical and possibly even correct.

You have two weeks to complete this assignment (rather than the normal one). This should let you not only get this code correct but also catch up with all the previous assignments so that you should end up with a working first half of your compiler when you are done.